NOTE: MAIN PROOF STARTS ON PAGE 4. THE FIRST 3 PAGES ARE JUST MY THOUGHTS AND INITIAL ATTEMPTS AND A DETAILED RECORD OF HOW THEY FAILED MISERABLY.

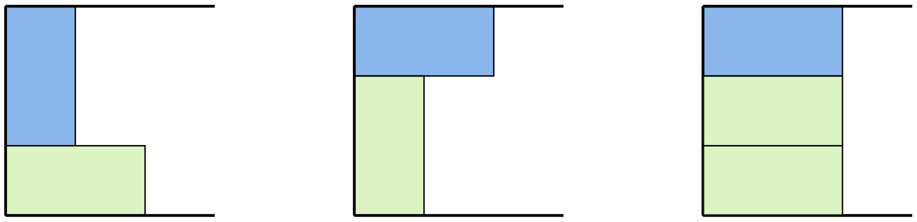
Chocolate

Give an algorithm with running time $f(k) \cdot p(n)$ to output the number of distinct legal tilings of a $k \times n$ hallway using 2×1 rectangular tiles, where $f(k)$ is allowed to be singly exponential and $p(n)$ polynomial.

Attempt 1, hints. For $\boldsymbol{k = 2}$, it is easy to see that

$$f(n) = f(n-1) + f(n-2), \tag{*}$$

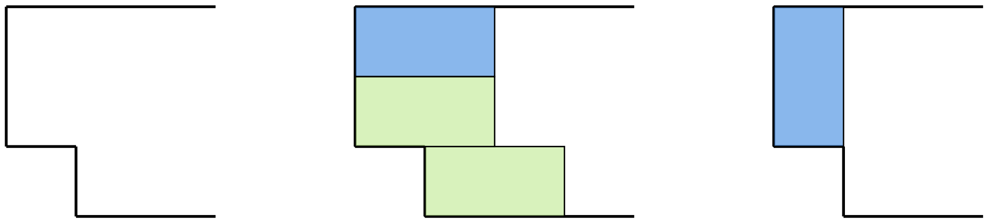
where $f(n)$ denotes the number of ways to legally tile a $2 \times n$ floor. Base cases: $f(0) = 0, f(1) = 1$, and $f(2) = 2$. Now we consider $\boldsymbol{k = 3}$. The topleft corner tile must be covered by some 2×1 rectangular tile, which can be either horizontal or vertical. We break this into different cases by fixing the blue rectangular tile and see where the green ones can be. This yields three possibilities.


NW Vertical NW Horizontal Triple Horizontal

A bit of abuse of notation here: let $f(n)$ again denote the number of legal $3 \times n$ tilings, and let $g(n)$ denote the number of legal $3 \times n$ tilings *with one corner removed.* The above gives

$$f(n) = f(n-2) + 2g(n-1).$$

Now we analyze the number of valid tilings in (a), removing the two rectangles already colored:


Original (a1) NW Horizontal (a2) NW Vertical

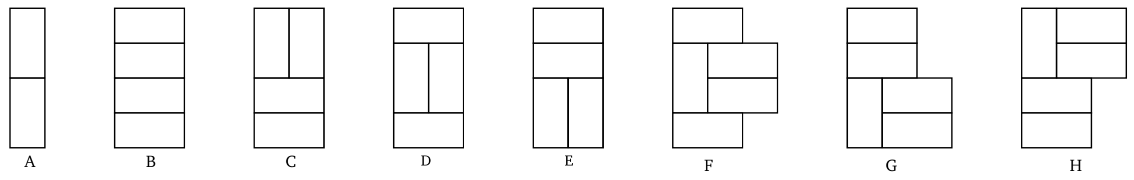
In particular, the blank region in (a1) has $g(n-3)$ valid tilings and that in (a2) has $f(n-2)$. Therefore,

$$f(n) = f(n-2) + 2(g(n-3) + f(n-2)) = 3f(n-2) + 2g(n-3).$$

We also know $f(n-2) = f(n-4) + 2g(n-3)$ from (1), so combining these,

$$f(n) = 4f(n-2) - f(n-2) + 2g(n-3) = 4f(n-2) - f(n-4). \tag{**}$$

Now we consider $\boldsymbol{k} = \mathbf{4}$. Below are the possible ways to tile the leftmost columns. Note that these initial layouts are mutually exclusive and cover all possibilities for the first two columns.



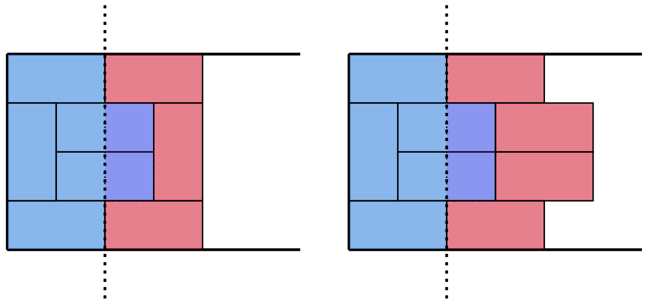
We again abuse notation, letting $f(n)$ denote the number of legal tilings of a $4 \times n$ floor. Also, for $\mathcal{S} \subset \{A, B, \ldots, H\}$, we define $f_{\mathcal{S}}(n)$ to be the number of ways to tile $4 \times n$, starting with any layout in $\mathcal{S}$. Since the layouts are mutually exclusive, this function is additive over disjoint sets, i.e., $f_{\mathcal{S}_1 \cup \mathcal{S}_2}(n) = f_{\mathcal{S}_1}(n) + f_{\mathcal{S}_2}(n)$ if $\mathcal{S}_1 \cap \mathcal{S}_2 = \varnothing$, and clearly f over $\{A, B, \ldots, H\}$ is just f.
It is clear that $f_{\{A\}}(n) = f(n-1)$ and that $f_{\{B,C,D,E\}} = 4f_{\{B\}}(n) = 4f(n-2)$. It remains to deal with the other three sets, F, G, and H.

- If we start with F, we can alternatively view the problem as the following:

  How many legal tilings exist for a $4 \times (n-2)$ hallway, assuming the middle two tiles in the first column (shown in purple) have already been covered?

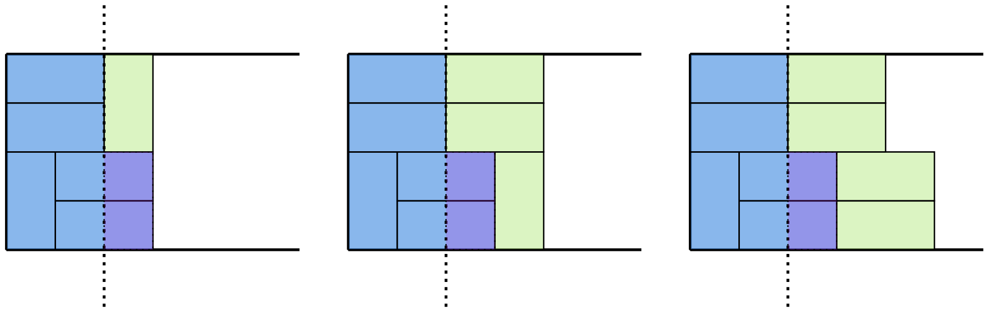
  That is, we consider a $4 \times (n-2)$ tiling problem with D sand F being the only available choices. Therefore,

$$f_{\{F\}}(n) = f_{\{D,F\}}(n-2). \tag{4F}$$



- If we start with G, we again have a $4 \times (n-2)$ tiling problem with the bottom two tiles of the first column already painted. Here, our available initial layouts are A, E, and G. Thus,

$$f_{\{G\}}(n) = f_{\{A,E,G\}}(n-2). \tag{4G}$$



- By an identical argument,
$$f_{\{H\}}(n) = f_{\{A,C,H\}}(n-2). \tag{4H}$$

Combining everything, we obtain the recurrence relation

$$\begin{aligned} f(n) &= f(n-1) + 4f(n-2) + f_{\{D,F\}}(n-2) + f_{\{A,E,G\}}(n-2) + f_{\{A,C,H\}}(n-2) \\ &= f(n-1) + 4f(n-2) + f(n-2) + f_{\{A\}}(n-2) - f_{\{B\}}(n-2) \\ &= f(n-1) + 5f(n-2) + f(n-3) - f(n-4). \end{aligned} \tag{***}$$

Now we consider $\boldsymbol{k} = \mathbf{5}$. Just kidding. Even the number of initial layouts is growing exponentially, and it quickly becomes impractical to enumerate and analyze case by case with the help of one auxilary function $f_{\mathcal{S}}$. Also, since my ignorance has prohibited me from spotting any relations between (*), (**), and (***) with respect to increasing k, I conclude that this is not the right path for me. Any further hints will be appreciated.

Dead proof below, ignore. See next page for the real one.

~~Attempt 2, checkerboards and permutations~~. Alternatively, we note that this question is closely related to checkerboards. In particular, if we treat the hallway as a huge checkboard, each 2×1 rectangle will cover exactly one white and one black tile. Given an $m \times n$ checkboard (assuming $2 \mid mn$ or otherwise no legal tiling exists), we can number each white tile by one distinct number in $\{1, \ldots, mn/2\}$ and similarly to black tiles. The placement of 2×1 rectangular tiles can be therefore thought as of a perfect matching between the white and black tiles, in particular, a permutation on $(1, 2, \ldots, mn/2)$. A valid tiling on the hallway therefore corresponds to a permutation $\sigma \in S_{mn/2}$ such that

$$\varphi(\sigma) := \prod_{i=1}^{mn/2} \mathbf{1}[\text{white tile } i \text{ is adjacent to black tile } \sigma(i)] = 1,$$

and our original goal is equivalent to calculating $\sum_{\sigma \in S_{mn/2}} \varphi(\sigma)$.
There will be an enormous room for optimizations, but below is an naïve algorithm that meets the complexity requirement.

- Convert hallway into checkerboard.
- Arbitrarily assign each white tile an integer in $[1, mn/2]$ without repetition. Do the same for black tiles. For example, going over tiles in each row, assign white 1, black 1, white 2, black 2, and so on.
- Maintain an array of vectors of size $mn/2$. For the j^{th} index of this array (assuming 1-indexed), fill a vector consisting of the indices of all black tiles that are adjacent to white tile j. Preferrably, ensure each vector is sorted.
  - Since there are $(m-1)(n-1)$ adjacent pairs in a $m \times n$ checkerboard, and each pair corresponds to exactly two entries among all vectors (one for the black tile, the other white), the total space occupied by this array of vectors is $\mathcal{O}(mn)$.
  - One white tile can be adjacent to at most four black tiles, and vice versa. This is the key to guaranteeing our singly exponential runtime.
- Brute-force examine all permutations using these adjacency info, using backtracking. We start with white tile 1, and we call the following recursive function. (I might have messed up something related to freeing occupied black tiles in the backtracking stage. I'll fix it when time permits. Main idea holds.)
  - Parameter: start with white tile ℓ.
  - While there are available adjacent neighboring black tiles:
    - Let i be the lowest index.
    - Try letting $\sigma(\ell) = i$.
    - If valid, i.e., no $\sigma(p) = i$ for $p < \ell$, keep this assignment.
      - If $\ell = n$, increase number of valid tilings by 1.
      - Otherwise, temporarily mark this black tile as occupied and recursivly call this function on white tile $\ell + 1$.
    - If invalid, continue.

Since each white tile has at most 4 neighboring black tiles, the algorithm above, albeit brute-force, examines at most $4^{mn/2}$ matchings between white and black tiles and outputs the number of permutations corresponding to valid tilings.

Main Solution

Attempt 3, algorithm ignoring closed form formula. Apparently, I would not easily give up, and with some extra hints, I was able to figure our an algorithm and test it on the recurrence relations (*), (**), and (***). The algorithm will take $\mathcal{O}(kn2^k)$ time and space, where $k < n$.

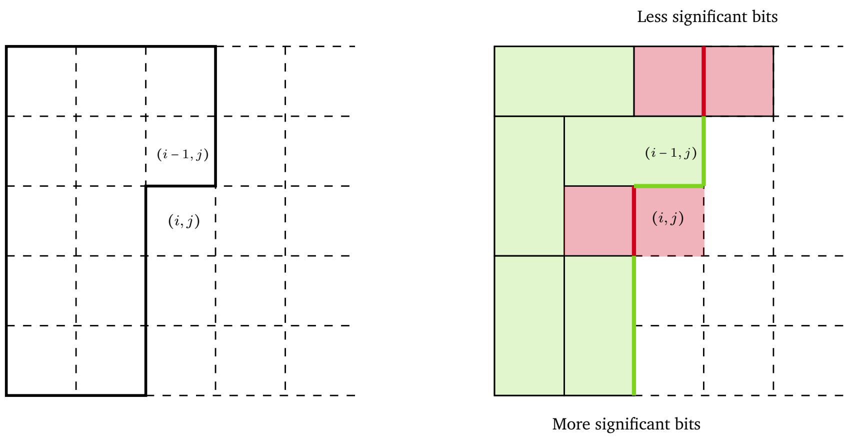
```python
def i_dont_know_how_to_count(k, n):
    if k * n % 2 == 1: return 0

    # create dp[][][], dimension k * n * 2^(n+1)
    # instead of dealing with 0-indexed arrays, I decide to simply pad it by making dp[][][] slightly larger. Index problems begone!
    # main idea: traverse through the hallway, and for each tile, keep track of all possible tilings of all previous tiles such that
      # (i) no 2x1 rectangles overlap
      # (ii) all previous tiles are covered, and
      # (iii) it does not matter if some 2x1 rectangles cover to-be-explored tiles, but they must lie within the hallway, i.e., not ouside the hallway's boundary
    # detailed explanation (e.g. choice of mask and update rules) below the algorithm

    max_mask = (1 << (k+1)) - 1
    dp = [[[0 for _ in range(1 << (k+2))] for _ in range(n+1)] for _ in range(k+1)] # slightly extra spaces
    dp[k][0][0] = 1 # base case: vacuously one way to tile a non-existent hallway

    for j in range(1, n+1):
        # the first for loop will execute every time we move on to a new column. In particular, it will carry all tiling information from tile[k][j-1] to tile[0][j] and prepare it for the first tile on column j, namely, tile[1][j] (recall 1-indexed arrays)
        for mask in range(max_mask+1):
            dp[0][j][mask << 1] += dp[k][j-1][mask]

        # this is how we traverse through most of the hallway, and the bulk of our DP algorithm. Main idea: decide whether and/or how to place the next rectangle given the current configuration, which is jointly decided by coordinates (i,j), and the mask.
        for i in range(1,k+1):
            for mask in range(max_mask+1):
                NE, SW = mask & (1 << (i-1)), mask & (1 << i)
                if NE and SW: continue
                if NE and not SW: dp[i][j][mask ^ (1 << (i-1))] += dp[i-1][j][mask]
                elif SW and not NE: dp[i][j][mask ^ (1 << i)] += dp[i-1][j][mask]
                else:
                    dp[i][j][mask ^ (1 << (i-1))] += dp[i-1][j][mask]
                    dp[i][j][mask ^ (1 << i)] += dp[i-1][j][mask]

    return dp[k][n][0] # done!
```

First, a graphical representation of how I traverse through each tile, what data I store, and how I define the mask. See below.

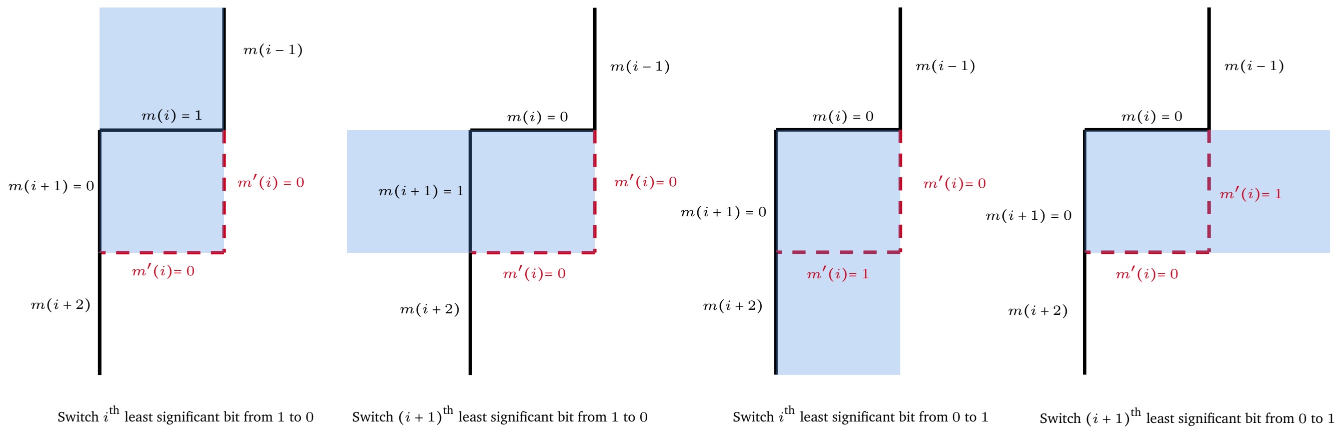
- At state (i, j), we will have traversed through the first $j - 1$ full columns, along with the first $i - 1$ tiles on the j^{th} column. Recall that everything is 1-indexed. On the figure below, $(i, j) = (3, 3)$.
- The particular boundary of interest is colored in green or red. In particular, it starts at the SE vertex of tile

$(k, j-1)$ and ends at the NE vertex of tile $(1, j)$. It is immedaite that this path has length $k+1$. It includes the west and north edges of tile (i, j).

- To define a mask corresponding to this particular tiling, we note that some part of this path has a 2×1 rectangular laid over it, whereas the other part does not. The former is colored in red and the latter in green.
- Naturally, we define our mask to be a $(k+1)$-bit binary number, with 1's corresponding to red and 0's to green, with the north endpoint being least significant and south most. It is clear that different masks correspond to different tilings. We use `dp[i][j][mask]` to store the number of computed tilings up to this point. For a more formal definition, this is the number of tilings such that at least at least half of each 2×1 rectangle is inside the "explored area."
- For example, the path and the tiling in the figure below correspond to the mask $(001001)_2$.



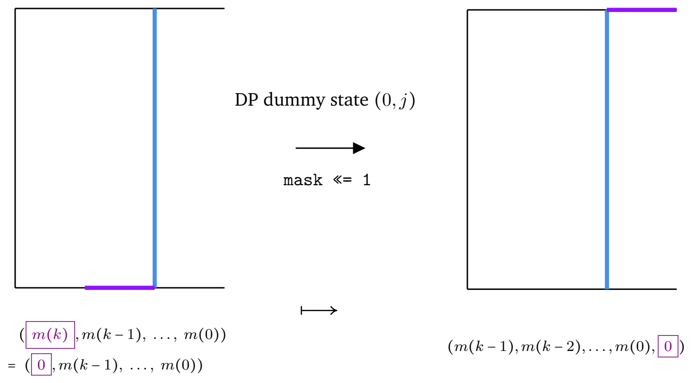
Now, the update rules for (i, j) with $i \neq k$ (i.e., not changing columns). For convenience, we denote the digits of $(k+1)$-bit mask by $m(k), m(k-1), \ldots, m(0)$, with $m(0)$ the least significant.



- We use variables `NW` and `SE` to indicate whether $m(i), m(i+1) = 1$, respectively. To do so, we need to check if the i^{th} (resp. $(i+1)^{\text{th}}$) least significant bits of `mask` is nonzero, as shown in line 24 using bitwise `AND` and left shift.

- It is impossible to have $m(i) = m(i+1) = 1$, as this implies tile (i,j) is both covered by a horitonzal rectangular tile (with $(i, j-1)$) and a vertical one (with $(i-1, j)$ assuming $j \geqslant 2$).
- If $m(i) = 1$ and $m(i+1) = 0$, this means tiles (i,j) is covered by a retangular tile along with $(i-1,j)$. Updating the boundary, replacing black $m(i), m(i+1)$ by the red dashed ones, we simply set $m'(i) = 0$, using bitwise `XOR` and left shift. We copy `dp[i-1][j][mask]` into `dp[i][j][new_mask]`, as the number of tilings corresponding to the new mask does not change.
- The case $m(i) = 0$ and $m(i+1) = 1$ is analogous.
- If $m(i) = m(i+1) = 0$, then tile (i,j) is not yet occupied. We can either place a horizontal rectangle with (i,j) being the left one, or a vertical one with (i,j) on top. Correspondingly, we either change modify $m'(i)$ or $m'(i+1)$, using almost identical techniques as before. We copy `dp[i-1][j][mask]` into both `dp[i][j][mask_new_horizontal]` and `dp[i][j][mask_new_vertical]`.

Finally, to switch to a new row, we note that the new mask is simply old mask with a right shift, as the two purple edges always corresponde to 0 — we allowed tilings to hit unexploreded areas, but *not* outside our hallway!


DP dummy state $(0, j)$
mask «= 1
$(\boxed{m(k)}, m(k-1), \ldots, m(0))$
$= (\boxed{0}, m(k-1), \ldots, m(0))$
$\longmapsto$
$(m(k-1), m(k-2), \ldots, m(0), \boxed{0})$

It should be clear that this algorithm has runtime $\mathcal{O}(n+1)(k+1)2^{(k+2)} = \mathcal{O}(nk2^k)$. I am, obviously, hoping that it is correct, so I plugged in value pairs $(k,n) \in \{2,3,4\} \times \{1,2,...,10\}$ and confirmed that they agree with (*), (**), and (***) in solution attempt #1.

	$n = 1$	2	3	4	5	6	7	8	9	10
$k = 2$	1	2	3	5	8	13	21	34	55	89
$k = 3$	0	3	0	11	0	41	0	153	0	571
$k = 4$	1	5	11	36	95	281	781	2245	6336	18061

So yes, either I screwed up the entire thing, or with probability $1 - \epsilon$ this algorithm is valid.